**Building a REST API with Mocked Database Interaction: Solution Document**

Introduction

This document outlines the solution for building a REST API with CRUD endpoints, searching, filtering, pagination, sorting, and implementing a retry and backoff mechanism for database write operations. The solution is implemented in a .NET Core Web API project using C#.

Solution Overview

Set Up Project: Created a .NET Core Web API project named EntityApi.

Defined Entity Models: Implemented the IEntity interface and its concrete implementation Entity, along with Address, Date, and Name classes representing the schema model.

Implemented Mocked Database Interaction Layer: Created a MockEntityService class to handle CRUD operations and data retrieval, with mocked data for testing purposes.

Implemented CRUD Endpoints: Implemented controllers and endpoints for creating, reading, updating, and deleting entities using the Entity service.

Implemented Searching and Filtering: Implemented endpoints to search for and filter entities based on specified criteria.

Implemented Pagination and Sorting: Implemented pagination and basic sorting capabilities for retrieving entities.

Implemented Retry and Backoff Mechanism: Implemented a retry and backoff mechanism in the MockEntityService class to handle transient failures during database write operations.

Reasoning Behind the Approach

Mocked Database Interaction: Used a simple in-memory list to mock database interaction, enabling easy testing and development without relying on an actual database.

Separation of Concerns: Followed the principle of separation of concerns by creating service classes to handle business logic and controllers to manage HTTP requests and responses.

Retry and Backoff Mechanism: Implemented a retry and backoff mechanism to enhance system robustness and handle transient failures gracefully. Chose a reasonable number of retry attempts and implemented an exponential backoff strategy to prevent overwhelming the database during failures.

Test-Driven Development: Used test cases to ensure the correctness of the implementation and verify the behavior of the retry and backoff mechanism in different failure scenarios.

Documentation: Provided detailed documentation and comments in the code to improve readability and facilitate understanding for future maintenance and collaboration.

Future Considerations

Database Integration: Replace the mocked database interaction layer with a real database integration for production use.

Authentication and Authorization: Implement authentication and authorization mechanisms to secure the API endpoints.

Error Handling: Enhance error handling to provide meaningful error messages and handle exceptions effectively.

Performance Optimization: Optimize performance by fine-tuning database queries, caching frequently accessed data, and implementing appropriate indexing.

Logging: Integrate logging mechanisms to record relevant information for debugging and monitoring purposes.

Conclusion

The implemented solution provides a robust and scalable REST API for managing entities with comprehensive functionality, including CRUD operations, searching, filtering, pagination, sorting, and resilience against transient failures through a retry and backoff mechanism. By following best practices and principles, the solution offers flexibility, maintainability, and extensibility for future enhancements and improvements.